Development of a Systematic Approach to Bottleneck Identification in UNIX systems

Development of a Systematic Approach to Bottleneck Identification in UNIX systems

Lori Park

TR-05-97

Computer Science Group
Harvard University
Cambridge, Massachusetts

Contents

1 Introduction **4**

2 Background **5**
- 2.1 Overview . . . 5
- 2.2 Computer Performance Evaluation . . . 5
- 2.3 Related Work . . . 6
- 2.4 Motivation . . . 6
- 2.5 Problem Statement . . . 7

3 Experimental Methods **8**
- 3.1 Overview . . . 8
- 3.2 FASCS System Architecture . . . 8
- 3.3 Tool Design . . . 9
  - 3.3.1 Tool Overview . . . 9
  - 3.3.2 Monitor Design . . . 10
  - 3.3.3 Further Design Considerations . . . 11

4 Discussion **15**
- 4.1 Overview . . . 15
- 4.2 Data Presentation . . . 15
- 4.3 Data Interpretation . . . 15
- 4.4 Tool Limitations/Future Work . . . 22

5 Conclusion **23**

6 Acknowledgements **24**

List of Figures

1 Diagram of FASCS Architecture 9
2 Kilobytes Transferred per Second vs. Time at Aggregation Levels of 10, 30, 300 12
3 Percent CPU Idle Time vs. Time at Aggregation Levels of 10, 30, 300 13
4 Kilobytes Transferred per Second vs. Time collected on mailtemp.fas from March 7 until March 15, 1997 16
5 Percent CPU Idle Time vs. Time collected on mailtemp.fas from March 7 until March 15, 1997 19

1 Introduction

As computers become more affordable, large computer systems like Harvard University's Faculty of Arts and Sciences Computer System (FASCS) are becoming commonplace. However, the price of the hardware and software is deceptively low and insignificant when compared to a system's operating costs, the bulk of which is spent on system performance management. According to an estimate by the Gartner Group the equipment costs of computer systems account for only 17% of the cost of owning a computer system. The remaining 83% goes to maintenance costs and performance management [5]. However, the performance problems faced by most users are more concrete than an expensive item in their company's budget. Poor computer performance is a problem every computer user has experienced. For many users, it is a common and recurring problem.

An ideal solution to computer system performance problem would be the development of an application that monitors the "health" of a computer system, notices when a problem arises, and fixes the problem – all without human intervention. This system should be highly scaleable (it should work for systems of 5, 50, or 50,000 machines) and it should be flexible (it should be able to recognize which system characteristics affect performance and adapt to any changes). Practically, a completely automated solution is nearly impossible because some performance problems can only be solved by installing more and/or faster equipment. However, in concept, a system that monitors system resources, detects and then diagnoses problems, and then administers a fix by either changing system parameters, terminating runaway processes, or notifying the system administrator of problems it is unable to solve (preferrably with a recommendation of how to solve problem based on the diagnosis) seems plausible.

Realization of the ideal solution described above seems to be the main emphasis of those studying computer system performance. People are working to deliver the ideal solution, yet no one has studied system behavior to make sure that the assumptions made about system behavior, upon which current techniques rely, are accurate. In this thesis, I suggest a method for characterizing "normal" system behavior. A clear understanding of system behavior may aid in the development of a solution to computer system performance problems.

This thesis is organized as follows. Chapter 2 gives background on the study of computer system performance, and then situates this thesis in the current state of performance management technology. Chapter 2 also motivates my work in the characterization of normal system behavior. Chapter 3 discusses the an approach to the characterization of normal system behavior. Chapter 4 provides a discussion of the results and recommendations for future work. Chapter 5 concludes the thesis.

2 Background

2.1 Overview

This chapter begins with a description of the study of computer system performance. Next is an overview of related work, motivation for the thesis, and a statement of the problem.

2.2 Computer Performance Evaluation

Unfortunately, the ideal solution to computer system performance problems described in Section 1 is far from realized. However, before presenting the current work in the field, the study of computer system performance problems and related terminolgy is more formally introduced.

The study of computer system performance problems emcompasses tasks such as *performance monitoring*, *performance management*, and *capacity planning* and is referred to as *computer performance evaluation* by Hellerstin [3].

Performance monitoring allows one to observe and record the behavior of a computer system by measuring the utilization levels of particular system *resources*. The resources of a system include the CPU, the I/O subsystem, the network subsystem, the (virtual) memory subsystem and the remote filesystem. Monitors may also perform very basic analysis of the data collected.

The characteristics measured are commonly called *performance metrics*. Examples of metrics include the number of bytes transferred to a disk per second, and the number of packets received per second. Optimally, metrics are chosen because they are accurate indicators of the "health" of the system. However, according to Hellerstein, metrics are often selected in a somewhat arbitrary manner resulting in metrics which are not always indicative of system health [1]. Accordingly, Hellerstein has developed a rule-based approach to performance metric selection [1]. Alternately, Loukides claims that there is already an established set of system metrics considered to be accurate indicators of system health [8].

After data is collected by the monitor, it is processed to reveal further information about the system. Hellerstein refers to this part of computer system evaluation as *performance management* [2]. There are three steps to resolving a performance problem. As defined by Hellerstein, these three steps are *problem detection*, *diagnosis*, and *treatment* [2]. Problem detection simply means knowing when a system is performing sub-optimally. Diagnosis begins after a performance problem has been detected. During diagnosis the data collected by the monitors is more closely analyzed to identify the cause of the problem, usually referred to as the *bottleneck*. After the source of the performance problem is identified, the system undergoes treatment. Treatment might entail the adjustment of system parameters by a system administrator or a performance management application.

At some point, adjustment of system parameters will no longer result in satisfactory performance improvement. At this point, more and/or faster equip-

ment (hardware and/or software) must be purchased. Capacity planning is the process by which one can quantitatively know that this point has arrived, and even predict when this point will occur again. Capacity planning also involves determining which hardware and software should be added to the system.

2.3 Related Work

There has been a considerable amount of work done on different aspects of computer performance evaluation. A significant amount of the related work concentrates on one or more of the three steps comprising performance management. A common approach involves developing a model of the system that is meant to predict the utilization levels of performance metrics. A system administrator, or expert system, can then compare that prediction to the actual utilization level of the metric. Theoretically, from this comparison one should be able to determine whether the utilization level of the particular metric is significantly higher or lower than predicted by the model. If the utilization level is unexpectedly high or low, there may be a performance problem. This is how problem dectection works in most systems. Further, the resource measured by that particular metric may be the cause of the performance problem (e.g., the bottleneck). After a diagnosis has been made, a treatment is administered. In most cases, diagnosis is much more complicated than described above. A thorough description of diagnosis and treatment is outside the scope of this thesis; however, both have been described in other works [6], [2], [3], [4].

2.4 Motivation

The current state of computer performance evaluation has been presented above. As far as I can tell from my review of the related work, the techniques described are based upon little knowledge of the nature of computer system behavior. Although performance monitors do observe the system, they have not been used to characterize the normal behavior of a computer system. Yet, it makes sense that before a performance evaluation technique is developed, the behavior of the system to which the technique is being applied should be understood. It is necessary that any assumptions about the system that the correctness of a given technique relies upon be provably true. Accordingly, in order to develop computer performance evaluation strategies that are truly effective, the behavior of computer systems be must characterized. This conclusion is supported by a study of LAN packet traffic. In a 1994 paper, Leland, et al. showed that Ethernet traffic exhibits behavior that is significatly different from the Poisson-like behavior assumed by most network models [7]. The discovery that the generally accepted Ethernet traffic model is seriously flawed had a serious implications on Ethernet LAN traffic modeling.

The ramifications of the work of Leland, et al. are still being realized. However, there is at least one important outcome: This new understanding of the nature of Ethernet traffic affects engineering and operations practices, and performance evaluation of high speed networks significantly [7].

Just as before the results of Leland, et al., the assumed Ethernet traffic behavior traffic had "very little to do with reality," the assumed behavior of computer systems may be similarly off base [7]. According to a 1994 paper by Hellerstein, all the expert systems developed to diagnose performance problems (of which he was aware) use at least one of *threshold analysis*, *Bottleneck analysis*, *what's different analysis*, and *correlation analysis* [3]. Hellerstein tested each one of these techniques by implementing an expert system which in turned applied each one of the techniques to a system with a known performance problem. In Hellerstein's experiment, none of the techniques correctly deduced the cause of the performance problem [3]. This may be because of assumptions the different techniques make (or do not make) about the nature of system behavior. For instance, each of the techniques considers data from all times during the day equally. This means that each of the methods will use the utilization of system resources at 3:00 a.m. in calculations made while diagnosing a problem at noon or no reference data at all is used. However, as far I can tell from reviewing related literature, no study had been done on whether the utilization of a resource at 3:00 a.m. really affects the utilization of a resource at noon, or if there is any relationship between data from two different points in time at all.

Since Hellerstein's 1994 paper [3], a technique employing time series analysis has been developed to diagnose performance problems. There have been two independent implementations of similar techniques, one by Hellerstein [4], and the other by Hoogenboom and Lepreau [6]. The interesting aspect of time series analysis is that the patterns in resource utilization levels, when the levels are considered over time, are identified. Therefore, this model acknowledges that, for a business establishment, resource usage is very likely to be similar, for example, from 11:30 a.m. until 1:00 p.m. (e.g. during the typical lunch hour). Another interesting aspect of this model is that it has been reasonably successful in two independent implementations. I believe that it is not a coincidence that the model which takes into consideration the nature of system behavior is the more successful of the five.

2.5 Problem Statement

When developing a technique for evaluating the performance of a system, the assumptions made about the system should be based on a firm understanding of the system's behavior. This statement is supported by the reults of Leland, et al. who found that the discovery of a better characterization for the behavior of LAN Ethernet traffic has significant implications for engineering design and performance evaluation of high speed networks. Therefore, in this thesis, I first propose a method for characterizing computer system behavior, and then describe my implementation of that method for a UNIX system. One goal is to show that there is a pattern in resource usage over time, and that a characterization of system behavior makes sense. Another goal is to suggest useful information one can learn from a charcterization of system data.

3 Experimental Methods

3.1 Overview

A tool to characterize normal system behavior was developed and applied to FASCS. This chapter describes the FASCS architecture and then explains the design of the tool. It ends by explaining the expected use of the tool.

3.2 FASCS System Architecture

FASCS is the distributed UNIX environment used by most of Harvard University and Harvard Extension School with about 16,000 users in total. In the last six months, the FASCS architecture has undergone major changes. The data presented in this thesis was collected from March 1, 1997 to March 18, 1997, during which time, the architecture did not change. The FASCS architecture for this period is shown in Figure 1. For the remainder of this thesis, any reference to FASCS will mean FASCS as it was from March 1 until March 18, 1997.

FASCS is composed of five major clusters of one or more machines connected by three subnetworks (commonly called subnets). These clusters are a general login cluster, an incoming mail cluster, a mailserver, a course cluster, and an user-directory cluster. The three subnets are 140.247.30.0, 140.247.31.0 and 140.247.33.0.

The subnets have different purposes. The 140.247.30.0 subnet is an external subnet for the login, incoming mail, and mailserver clusters. It has a 100 Mbps, full duplex connection. The 140.247.31.0 subnet is a subnet for the course cluster. The subnet 140.247.33.0 is an internal subnet for communication between the login, incoming mail, mailserver, and user-directory clusters. Both the 140.247.30.0 and140.247.31.0 have 10 Mbs, half duplex connections.

There are three general login servers which are primarily for email, lynx, news, and gopher, although in most cases, all services are available on all machines. These three login servers are named login1.fas, login2.fas, and login3.fas. The login servers all are on both the 140.247.30.0 and 140.247.33.0 subnets, and all have 512 Mb of RAM. The machine login2.fas was monitored for this thesis.

There are also three machines in the incoming mail cluster. These machines are called smtp1.fas, smtp2.fas, and smtp3.fas because they run the Simple Mail Transfer Protocol (SMTP) which receives and processes incoming mail. For the rest of this thesis, the incoming mail cluster will be referred to as the SMTP cluster. The SMTP servers all are on both the 140.247.30.0 and 140.247.33.0 subnets, and all have 256 Mb of RAM. The machine smtp2.fas was monitored for this thesis.

The mailserver, called mailtemp.fas, is in a cluster by itself. Additionally, it has a direct, SCSI connection to a Raid 5 array which stores all the user mail boxes. Besides as acting as a mailserver, the mailtemp.fas also provides the service for the Network File System (NFS), Post Office Protocol (POP), and Internet Messaging Access Protocol (IMAP). The mailserver is on all three subnets and has 512 Mb of RAM. This machine was monitored for this thesis.

The course cluster consists of six machines that provide the services necessary for coursework such as compilers, netscape, and statistical programs. The course servers all are on the 140.247.31.0 subnet and have from 16 Mb to 128 Mb of RAM. This cluster is not discussed in this thesis.

The user-directory cluster, is only one machine, a Network Appliance Box (NAB). The NAB stores all the user directories. It is on all three subnets and has 128 Mb of RAM. This cluster is not discussed in this thesis.

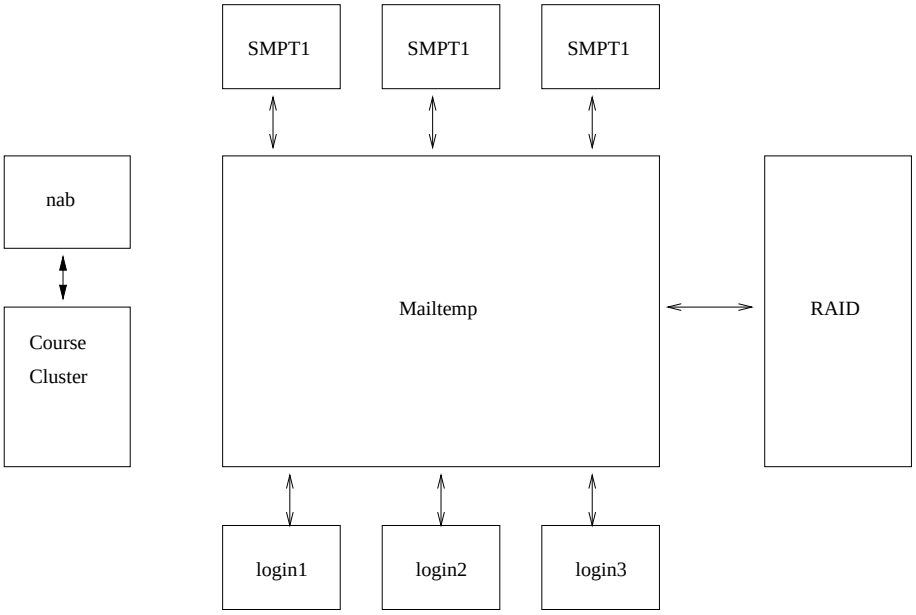


Figure 1: Diagram of FASCS Architecture

3.3 Tool Design

3.3.1 Tool Overview

A goal of this thesis is the characterization of normal system behavior, so monitors were designed to run continuously on FASCS machines. The monitors collect resource utilization information from the system which is then analyzed in order to determine the normal system behavior.

At a high level, the tool works by running on each machine under study and continuously collecting resource utilization information from the standard UNIX utilities such as *iostat*, *vmstat*, *uptime*, and *netstat*. The output from these utilities is parsed by scripts and graphed using Gnuplot. The graphs are stored by date for future viewing.

3.3.2 Monitor Design

The first step in the development of system monitors is deciding which system resources to monitor. As mentioned in Section 2.2, the resources of a UNIX system can be categorized by one of the UNIX subsystems: the I/O subsystem, the network subsystem, the (virtual) memory subsystem, and the remote filesystem which is the Network File System (NFS) in the case of FASCS. Loukides describes a set of performance metrics [8] many of which are monitored by my system. Additionally, UNIX has a set of built-in performance monitors (referred to as standard UNIX utilities in this thesis) that monitor the resources considered to be important by the developers of UNIX. The standard utilities are somewhat cumbersome for general usage as they must be invoked manually by the system adminstrator, and customization of the output is limited so it is easy to produce much more data than is necessary, or useful.

My monitors look at all the subsystems enumerated above. Therefore, it was necessary to find a way to extract information about each of the subsystems from the system. It is possible to extract data about each of these subsystems directly from the UNIX kernel. By doing this, I could retrieve the data in a format of my own design. This method is referred to as *kernel tracing* and requires that the person running the traces have super-user status on the system. Understandably, I was not given super-user status on FASCS so I had to choose a method of collecting data that did not require kernel access. Therefore, I had to use the standard UNIX utilites mentioned above which can be called at any time intervals. My monitors call the following standard UNIX utilities: *iostat*, *vmstat*, *uptime*, *netstat*, and *nfsstat*. A description of each follows. A more thorough description of the standard utilities can be found in the Digital UNIX documentation [9] and in the Digital UNIX *man* pages.

Iostat monitors the I/O subsystem providing information about the disks servicing each machine. Specifically, *iostat* reports the average number of transfers per second (tps), and the average number of kilobytes transferred per second (bps).

The standard UNIX utility *vmstat* lets one report usage statistics on the virtual memory system as well including the number of fork() system calls, the number of free pages, the number of pages paged in, the number of pages paged out, separate statistics on the number of processes running, waiting, and blocked, and the percentages of time the CPU spends idle, executing system code and executing user code. *Uptime* is closely related to *vmstat*. Most importly, *uptime* monitors the current load, and the number of users.

Two utilities monitor the network subsystem. *Netstat* reports the number of incoming and the number of outgoing packets as well as the number of collisions and dropped packets, the amount of memory dedicated to the network, and per-protocol breakdowns of the number of incoming and outgoing packets, and errors. *Nfsstat* reports both the client and server statistics for NFS.

The monitors collect data at two minute intervals and are scheduled by *cron*, a UNIX utility which schedules commands to be run at user-specified intervals. This collection interval is a design constraint imposed by the FASCS

administrators who were concerned that any smaller interval would place too much load on the system.

Now, if the quality of the data collected depends on the size of the collection interval, and if two minutes is too large an interval, then my system would be useless when collecting data at two minute intervals. In order to show that the chosen collection interval does not affect the quality of the data collected, it is necessary to show that the data collected are self-similar. A thorough discussion of self-similarity in data is outside the scope of this thesis; however, the property of self-similarity is discussed thoroughly by Leland, et al. [7]. Self-similarity can be shown by graphing the data verses time for each metric, and then by aggregating that data over increasingly large time intervals. If data is self-similar, it's distribution will not change with the time scale. Data that are not self-similar will smooth out as the aggregation increases.

To test whether my data is self-similar, I collected data (for one day) at 15 second intervals. Then I graphed the data over time with an aggregation level of 10, then 30, and then 300 times the original time interval. Figure 2 shows the aggregated graphs for the kilobytes transferred per second to the Raid 5 versus time, and Figure 3 shows the aggregated graphs for the percentage of CPU idle time versus time. From inspection, one can see that although there are fewer observations, the distributions remain regardless of the level of aggregation.

To gain further confidence in the validity of the two minute collection intervals, I should mathematically prove that the data is self-similar. In order to do this, I should calculate the Hurst parameter for each set of data at increasing levels of aggregation. The Hurst parameter, as described by Leland, et al., gives a measure for how self-similar data is [7].

After one day of data is collected, it is parsed. Then each performance metric is graphed versus time using gnuplot. This entire process has been automated.

This tool will work on any UNIX system. The machines monitored are running Digital UNIX 4.0. The format of the output of its standard utilities differs from the output of other members of the UNIX family. However, only a few simple scripts must be rewritten in order port this tool.

3.3.3 Further Design Considerations

The tool is meant to be very easy to use. At this time, a graph viewer that allows a user to view graphs of user defined intervals has not been implemented, but doing so would be fairly trivial. However, the design is one that allows a user to gain a strong knowledge of his or her system with little effort or confusion. Additionally, this tool was designed in such a way that a high level of mathematical sophistication or customization is not required. At the heart of the design is the idea each performance metric, will exhibit a daily (then weekly, monthly, etc.) usage pattern. Perhaps weekdays all have the same pattern, and weekends all have the same pattern. Similarly, vacation months all might have the same pattern, and term-time months might all have the same pattern. I designed the system thinking that it would be possible to characterize the normal system behavior by looking at the patterns that result

Figure 2: Kilobytes Transferred per Second vs. Time at Aggregation Levels of 10, 30, 300

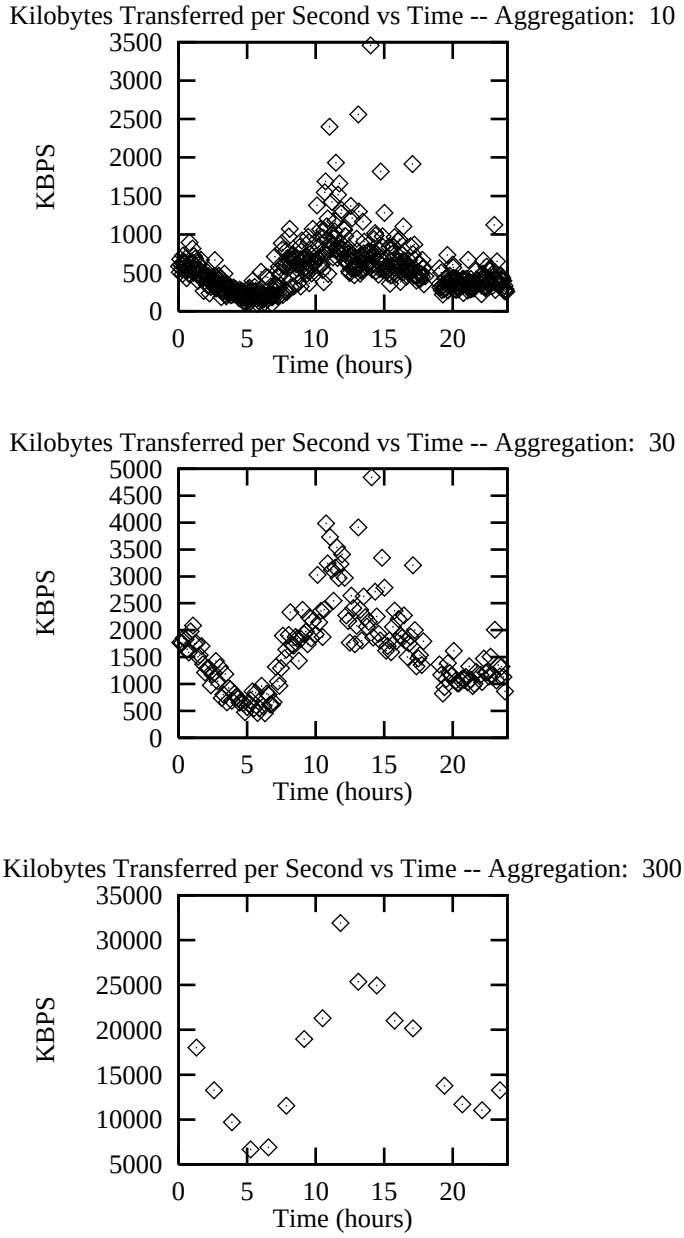
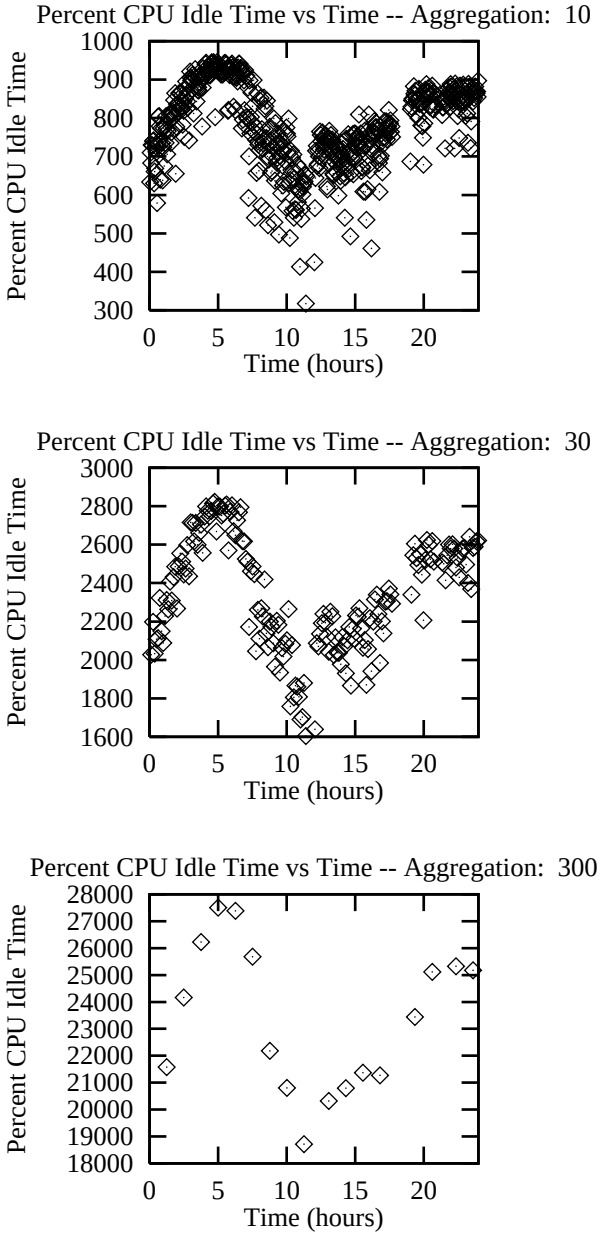


Figure 3: Percent CPU Idle Time vs. Time at Aggregation Levels of 10, 30, 300



when a metric is graphed versus time and comparing those patterns with the patterns from the same time in previous days, weeks or months (whichever points in time have similar patterns). Following this line of reasoning, one might find that weekdays have very similar patterns. Now, if some Wednesday a user notices that disk usage is significantly higher that it was the day before at the same time, the user could look at the last, for example, five weekdays to see whether the suspect behavior is indeed abnormal. Especially the user would want to look at the previous Wednesdays to see whether the high level of disk usage is normal for Wednesdays. From this information, the user could make a reasonable prediction about whether or not the behavior is abnormal. There is further analysis that could be done by the user; however the analysis described is enough to show the simplicity of the system.

4 Discussion

4.1 Overview

This section shows graphs generated by the tool, and then discusses how the information provided by the tool is useful for system administrators. Next limitations of the tool and suggestions for future work.

4.2 Data Presentation

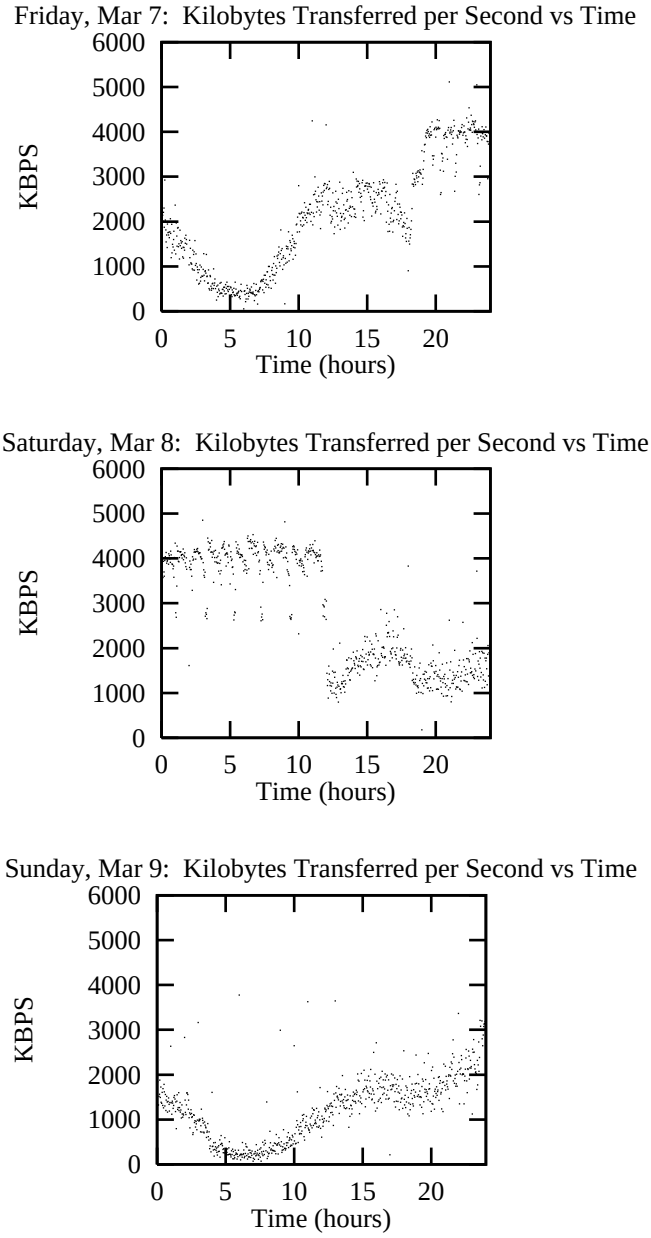
Figures 4 and 5 contain graphs of approximately a week of data that was collected by the monitors on mailtemp.fas. Because of space constraints, I am only showing two performance metrics, "Kilobytes Transferred Per Second vs. Time" (BPS) from the Raid 5 array, and the "Percent CPU Idle Time vs. Time" (CPU idle time). Although all the cllected data that I viewed had a distinct pattern across weekdays and across weekend-days (these are not completely precise pattern groups, but close), I chose BPS because of a very distinct in usage pattern for late Friday night and early Saturday. I chose CPU idle time because it indicates possible a FASCS performance problem.

4.3 Data Interpretation

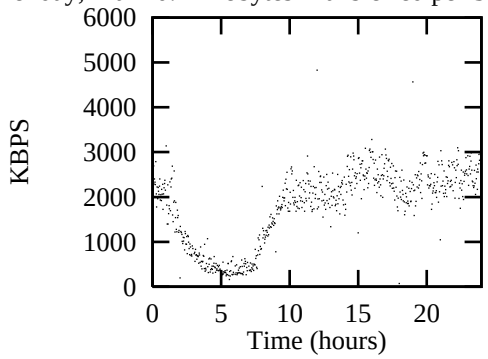
These graphs give an interesting characterization of normal system behavior. By visual inspection of the BPS graphs in Figure 4, one can see that on most days the disk usage has the a similar pattern. However, late on Friday and early on Saturday morning, there is a huge burst of sustained disk activity. This burst has occurred in all the weeks I have analyzed, and show up on both Saturdays in Figure 4. Visual inspection of the system over time allows one to see gain a real understanding of resource usage patterns leading to a characterization of system behavior. One may also observe unexpected bursts of activity or unusually high utilization levels that may signify a problem. For example, from the graphs in Figure 5, one can see that the percentage of time the CPU spends idle is consistently above 50%. This is an unusually high value for this metric and can be indicative of a memory bottleneck, an I/O bottleneck, or an extremely underutilized CPU. This tool helps a user recognize the abnormal patterns in behavior (as well as the normal patterns) so a user may be able to identify two or more separate problem behaviors if there are two or more different problem patterns. However, in a purely value driven approach, two different problem behaviors with similar utilization levels for the bottlneck resources, may be classified as the same problem. Further, this tool may aid in bottleneck identification. If a system is behaving abnormally because the value(s) of one or more of the metrics is significantly higher or lower than the normal behavior, then there is a possiblity that the resource exhibiting the abnormal behavior is the bottleneck resource.

Additionally one can observe which levels of utilization must be supported for a given resource. For instance, by just by looking at Figure 4, it is clear that

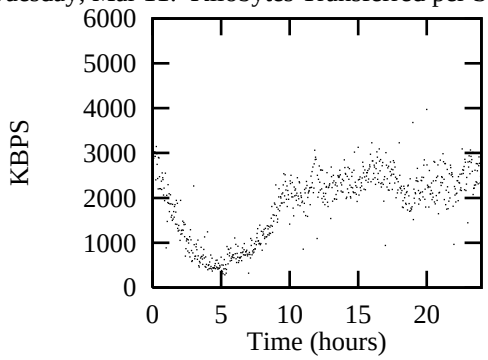
Figure 4: Kilobytes Transferred per Second vs. Time collected on mailtemp.fas from March 7 until March 15, 1997



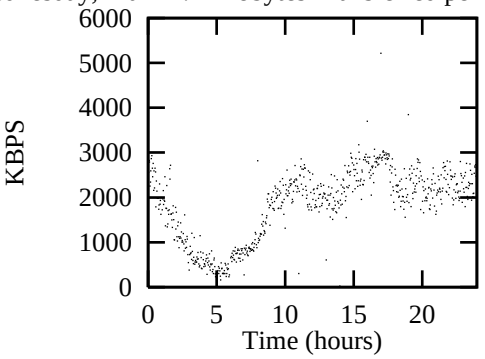
Monday, Mar 10: Kilobytes Transferred per Second vs Time



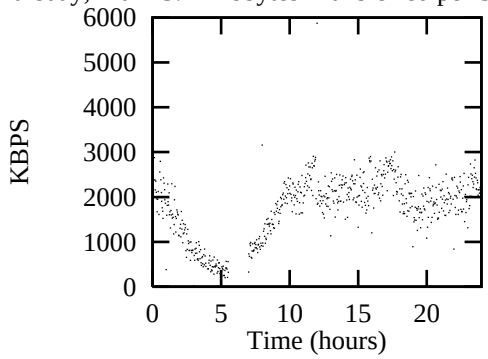
Tuesday, Mar 11: Kilobytes Transferred per Second vs Time



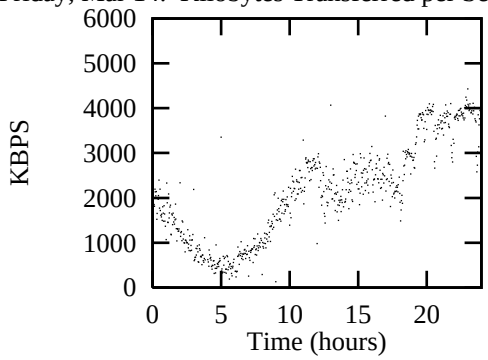
Wednesday, Mar 12: Kilobytes Transferred per Second vs Time



Thursday, Mar 13: Kilobytes Transferred per Second vs Time



Friday, Mar 14: Kilobytes Transferred per Second vs Time



Saturday, Mar 15: Kilobytes Transferred per Second vs Time

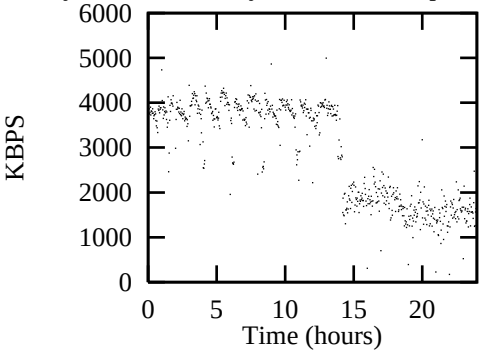
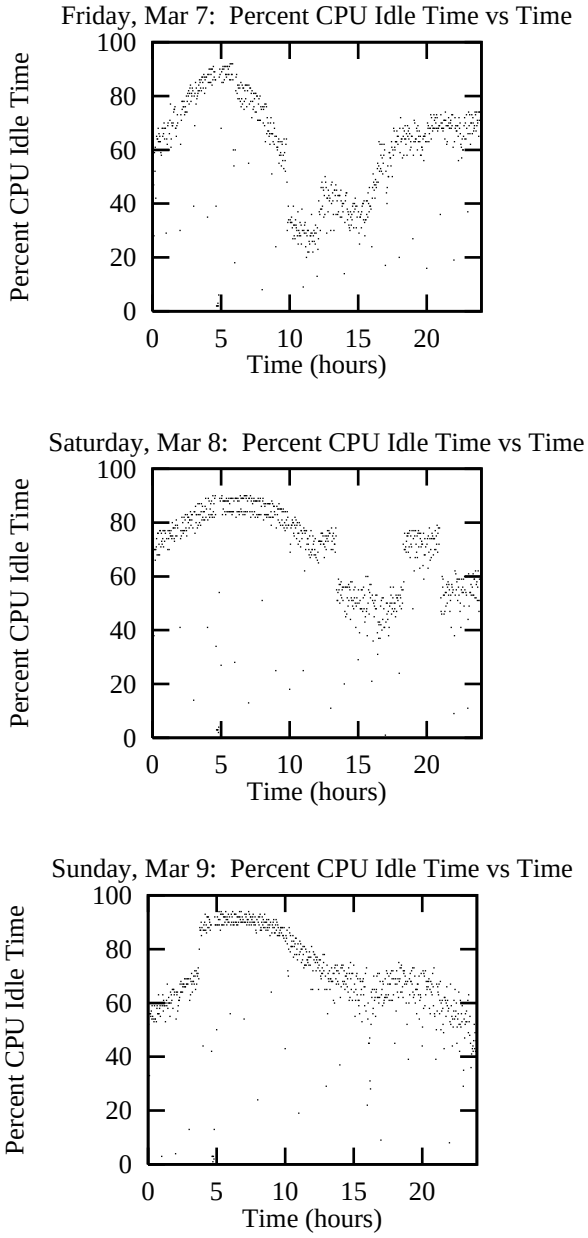
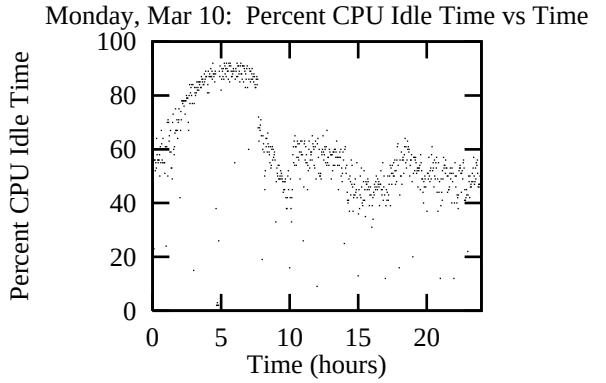
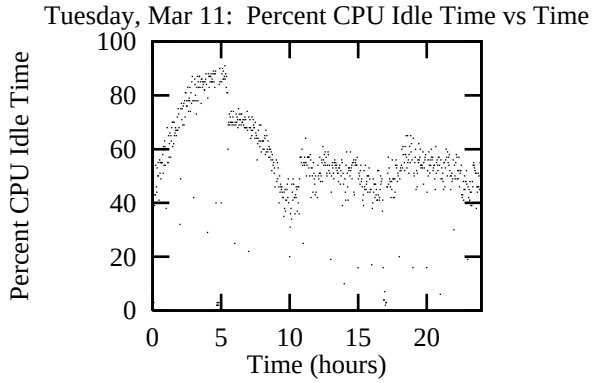
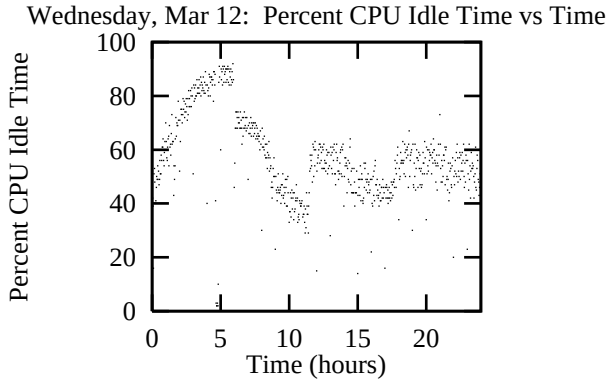


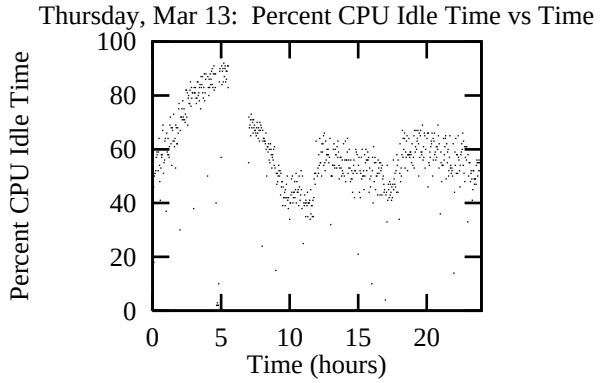
Figure 5: Percent CPU Idle Time vs. Time collected on mailtemp.fas from March 7 until March 15, 1997

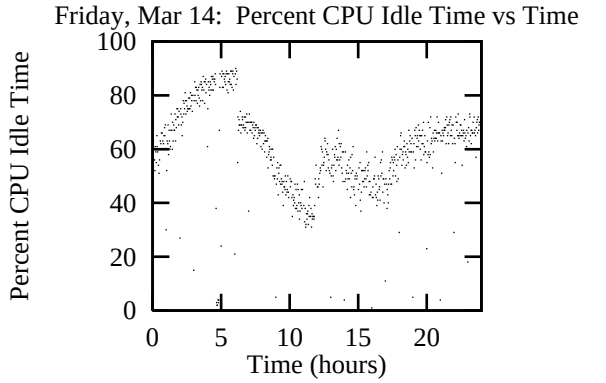


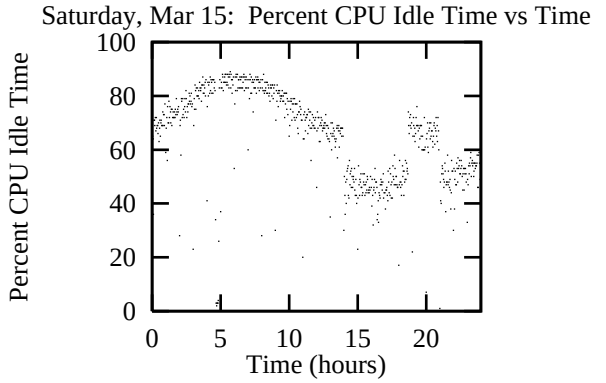

Monday, Mar 10: Percent CPU Idle Time vs Time
Percent CPU Idle Time
100
80
60
40
20
0
0
5
10
15
20
Time (hours)


Tuesday, Mar 11: Percent CPU Idle Time vs Time
Percent CPU Idle Time
100
80
60
40
20
0
0
5
10
15
20
Time (hours)


Wednesday, Mar 12: Percent CPU Idle Time vs Time
Percent CPU Idle Time
100
80
60
40
20
0
0
5
10
15
20
Time (hours)


Thursday, Mar 13: Percent CPU Idle Time vs Time
Percent CPU Idle Time
100
80
60
40
20
0
0
5
10
15
20
Time (hours)


Friday, Mar 14: Percent CPU Idle Time vs Time
Percent CPU Idle Time
100
80
60
40
20
0
0
5
10
15
20
Time (hours)


Saturday, Mar 15: Percent CPU Idle Time vs Time
Percent CPU Idle Time
100
80
60
40
20
0
0
5
10
15
20
Time (hours)

mailtemp.fas needs to be able to access it's disks at a rate of at least 3000 to 4000 kilobytes per second.

Further, by knowing the normal behavior of one's system, one can make well informed choices about system upgrades in two ways. First, from Figure 5, one can see that CPU is idle for a large percentage of every day. According to Loukides, the CPU idle time should be between 1% and 10% [8]. On mailtemp.fas, the CPU idle time is consistently above 50%. Just recently, significant upgrades have been made to FASCS, including machines with significanly faster CPUs. Had the FASCS administrators had this characterization of FASCS's normal behavior, they may have put more money in another resource and less into the CPU.

Knowing a system's normal behavior may also help in capacity planning. For instance, when the patterns of the days have the same curve as the patterns from three months ago, but the magnitudes of the utilizations become greater and greater (i.e. the curves are shifted vertically) as months pass, it may mean that new, faster and/or more resources are necessary.

4.4 Tool Limitations/Future Work

One clear limitation of this tool is that its fairly un-scaleable. The overhead to run the monitors and parse the data is rather high (although it has not been quantitfied). Further, the amount of data storage required for one machine after only a month is on average about 40 megabytes, which may be inhibitive. In fact, Hellerstein described this method as unmanageable in . One improvement would be to extract the data directly from the kernel, instead of using the standard utilities and parsing them. If this change were made, the collection and parsing process would be much more manageable with significantly less overhead.

Another severe limitation is that while this tool does give a user a clear idea of the normal system behavior, it does not give any method of determining when behavior is abnormal enough to be considered a performance problem. Developing a method to decipher when behavior is abnormal enough to be of concern is a problem for further study.

Further, a quantitative method for determining the correlation between two patterns is another problem for future study. Currently, only the user's judgement decides when patterns are the same.

Yet another limitation is that if the system configuration changes, all the collected data is no longer valid, and a new collection cycle must begin. In frequently changing systems, it may not be worth it to use the tool since one main benefit the tool provides is a comprehensive system history from which a characterization of normal system behavior can be derived. Without a significant history, the tool is not as useful.

Other future work could include the characterization of abnormal system behavior by studying the inter-relationships between resources so that abnormal combinations can be discovered. For instance, if both disk usage and CPU idle

time skyrocket, there may be a different problem then if just CPU idle time or just disk usage skyrockets.

5 Conclusion

For this thesis, a tool that does give a clear characterization of UNIX system behavior, and that may aid in the identification of system bottlenecks, and in capacity planning was developed. However, the collection mechanism is unwieldly and the system, as is, is not highly-scaleable.

6 Acknowledgements

Special Thanks to:

Professor Margo Seltzer, Professor H.T. Kung, Professor Michael Smith, abhi shelat, Sandra Batista, Nailah Robinson, Joe Marks, Mark Gaynor, Chris Small, Dan Ellard, and my thesis buddies.

References

[1] Hellerstein, Joseph L. *An approach to selecting metrics for detecting performance problems in information systems.* IBM RC 20221 IBM Research Center. Yorktown Heights, NY. Oct. 1995.

[2] Hellerstein, Joseph L. *A comparison of techniques for diagnosing performance problems in information systems: case study and analytic models.* IBM RC 19708 IBM Research Center. Yorktown Heights, NY. Aug. 1994.

[3] Hellerstein, Joseph. *How expert is your expert system for performance management?* IBM RC 19562 IBM Research Center. Yorktown Heights, NY. May 1994.

[4] Hellerstein, Joseph L. *An introduction to modeling dynamic behavior with time series analysis.* IBM RC 18893 IBM Research Center. Yorktown Heights, NY. May 1993.

[5] Hellerstein, Joseph L.; Calo, Seraphin B.; White, William W. *Management of complex information systems.* IBM RC 19689 IBM Research Center. Yorktown Heights, NY. Aug. 1994.

[6] Hoogenboom, Peter; Lepreau, Jay. *Computer System Performance Problem Detection Using Time Series Models.* 1993 Summer USENIX Conference(June 1993), pp 15-31.

[7] Leland, W.E.; Taqqu, M.S.; Willinger, W.; Wilson, D.V. *On The Self- similar Nature of Ethernet Traffic.* IEEE/ACM Transactions on Networking, February 1994, Vol 2, Number 1.

[8] Loukides, Mike. *System Performance Tuning.* O'Reilly & Associates, Inc., 1990.

[9] *Digital UNIX. System Tuning and Performance Management.* Digital Equipment Corporation, March 1996.